

A Short Note on Improved Logic Circuits in a Hexagonal Minesweeper

Seunghoon Lee

Department of Mathematics, Seoul National University*
galaxybp@snu.ac.kr

February 2, 2016

Abstract

This paper aims to present an advanced version of PP-hardness proof of Minesweeper by Bondt [1]. The advancement includes improved Minesweeper configurations for ‘logic circuits’ in a hexagonal Minesweeper. To do so, I demonstrate logical uncertainty in Minesweeper, which ironically allows a possibility to make some Boolean operators.

The fact that existing hexagonal logic circuits did not clearly distinguish the *true* and *false* signal needs an improved form of a hexagonal wire. I introduce new forms of logic circuits such as NOT, AND, OR gates, a curve and a splitter of wires. Moreover, these new logic circuits complement Bondt’s [1] proof for PP-hardness of Minesweeper by giving a new figure.

Keywords: Boolean circuit, PP-hard, NP-complete, Logic circuit.

1 Introduction

Every computer user in the world must, at least once, have played this game: Minesweeper. If anyone ever witnessed the yellow circle smiling with its sunglasses on, she or he may find this research interesting. A normal Minesweeper is played on a square grid, each compartment is enclosed by eight neighborhoods except on the border of the grid. When we click any compartment on the grid, a number between 0 and 8 appears showing the number of mines around the point one clicked. Of course, if the clicked point was exactly on a mine, then the game is over. The goal of Minesweeper is to find and check all compartments, which contain mines. There are many strategies to win the game, but existing strategies do not provide a perfect logic to beat the game. The following example shows this uncertainty.

*Currently on a leave of absence for the mandatory military service.

Example 1 (Uncertainty on 9×9 board). During Minesweeper gameplay in novice mode which consists of a 9×9 board, let us be given the situation as figure 1(a).

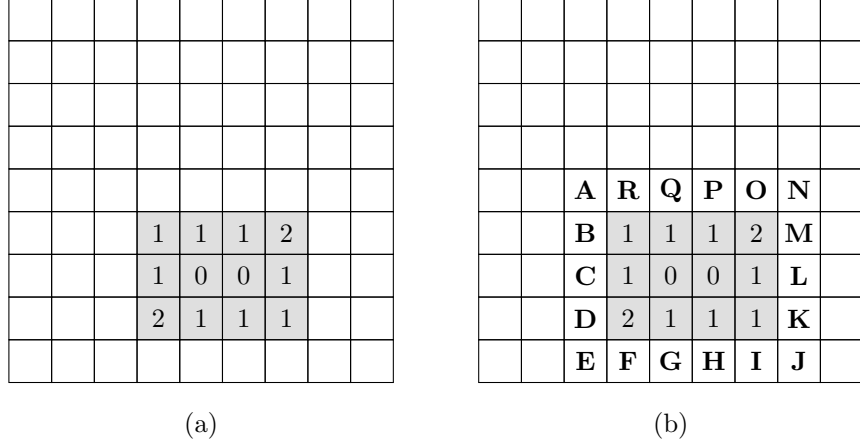


Figure 1: An example that shows uncertainty of Minesweeper

At first glance, it is difficult to determine the locations of mines. Once we allocate names from **A** to **R** as shown in (b). Then, there are 30 different possible locations for the mines. For example, if we set **B** as a mine, then **C, D, A, R, Q** should not be mines. Therefore, **P** should be a mine; since two of **E, F, G**, one of **F, G, H**, and one of **G, H, I** should be a mine, we conclude that **E** is a mine. Next choice is either **F** or **G**, if **F** is a mine, then automatically so are **I** and **M**. All possible arrangements of mines are given in the table 1¹:

Table 1: 30 different possibilities

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	A
1	○	×	×	○	○	×	×	○	×	×	×	○	×	×	○	×	×	×
2	○	×	×	○	×	○	×	×	○	×	×	○	×	×	○	×	×	×
3	○	×	×	○	×	○	×	×	×	○	×	×	○	×	○	×	×	×
4	○	×	×	○	×	○	×	×	×	×	×	○	×	×	○	×	×	×
5	×	○	×	○	×	×	○	×	×	×	×	○	×	×	○	×	×	×
6	×	○	×	×	○	×	×	○	×	×	×	○	×	×	○	×	×	×
7	×	○	×	×	×	○	×	×	○	×	×	○	×	×	○	×	×	×
8	×	○	×	×	×	○	×	×	×	○	×	×	○	×	○	×	×	×
9	×	○	×	×	×	○	×	×	×	×	○	×	×	×	○	×	×	×
10	×	×	○	○	×	×	○	×	×	×	×	○	○	×	×	○	×	×
11	×	×	○	○	×	×	○	×	×	×	×	○	×	○	×	×	○	×
12	×	×	○	○	×	×	○	×	×	×	×	○	×	○	×	×	×	○

¹○ means identified as a mine, and × means there is no mine.

Table 1: 30 different possibilities

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	A
13	×	×	○	○	×	×	○	×	×	×	×	○	×	×	○	×	×	○
14	×	×	○	×	○	×	×	○	×	×	×	○	○	×	×	○	×	×
15	×	×	○	×	○	×	×	○	×	×	×	○	×	○	×	×	○	×
16	×	×	○	×	○	×	×	○	×	×	×	○	×	○	×	×	×	○
17	×	×	○	×	○	×	×	○	×	×	×	○	×	×	○	×	×	○
18	×	×	○	×	×	○	×	×	○	×	×	○	○	×	×	○	×	×
19	×	×	○	×	×	○	×	×	○	×	×	○	×	○	×	×	○	×
20	×	×	○	×	×	○	×	×	○	×	×	○	×	○	×	×	×	○
21	×	×	○	×	×	○	×	×	○	×	×	○	×	×	○	×	×	○
22	×	×	○	×	×	○	×	×	×	○	×	×	○	○	×	×	○	×
23	×	×	○	×	×	○	×	×	×	○	×	×	○	○	×	×	×	○
24	×	×	○	×	×	○	×	×	×	○	×	×	○	×	○	×	×	○
25	×	×	○	×	×	○	×	×	×	×	○	×	○	○	×	×	○	×
26	×	×	○	×	×	○	×	×	×	×	○	×	○	○	×	×	×	○
27	×	×	○	×	×	○	×	×	×	×	○	×	○	×	○	×	×	○
28	×	×	○	×	×	○	×	×	×	×	○	×	×	○	×	×	○	×
29	×	×	○	×	×	○	×	×	×	×	○	×	×	○	×	×	×	○
30	×	×	○	×	×	○	×	×	×	×	○	×	×	×	○	×	×	○

Furthermore, when we check each column(**A** to **R**), we can easily find out that no column contains neither only ○ nor only ×; it follows that we cannot go even one step further logically. \square

Using this uncertainty, Kaye [2] made some Minesweeper configurations for ‘logic circuits’, and he proved that Minesweeper is NP-complete. Kaye [2] designated wires carrying either *true* or *false*; and using these wires he made several logic circuits such as NOT, AND, OR, XOR gates, etc.

In this paper, I apply this concept to a hexagonal grid. In section 3, I define a hexagonal wire and some logic circuits². Consequently, this improves Bondt’s [1] computational components on a hexagonal Minesweeper.

²I introduced this concept in a science essay that won President Science scholarship, Republic of Korea in 2004.

2 Computational components in a normal Minesweeper

First, I will review Kaye's [2] work on Minesweeper configurations. In Figure 2, Kaye [2] demonstrated a 'wire' that conducts x . We can easily figure out that either all x 's are 1 and all x' 's are 0 or vice versa. It is natural to say the Boolean values *negate* or *confirm* with x 's are 0 or 1 respectively.

$x \longrightarrow$																	
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
...	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
...	x	1	x'	x	1	x'	x	1	x'	x	1	x'	x	1	x'	x	...
...	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

Figure 2: A Wire on a normal Minesweeper

We also need to bend wires and to make a splitter to duplicate wires as Figure 3. From now on, the gray circles in the nodes indicate mines those have been identified.

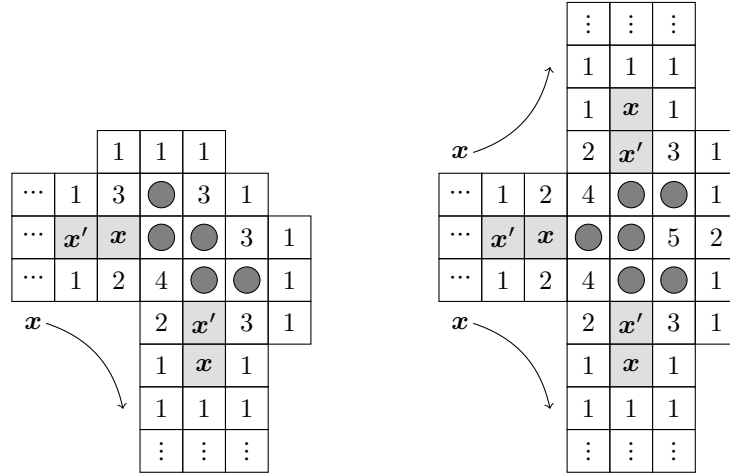


Figure 3: Curve and splitter of a wire (adopted from Bondt [1])

Using these basic components, existing studies propose some large computational components such as NOT, AND and OR gate. NOT and AND gates are created by Richard Kaye [2], and the OR gate is made by Stefan [3].

$x \longrightarrow$						1	1	1	$x' \longrightarrow$					
...	1	1	1	1	1	2	●	2	1	1	1	1	1	...
...	x'	x	1	x'	x	3	x'	3	x	x'	1	x	x'	...
...	1	1	1	1	1	2	●	2	1	1	1	1	1	...
						1	1	1						

Figure 4: A NOT gate on a normal Minesweeper

u	⋮	⋮	⋮																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	1	1	1																	1	1	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
	1	u'	1																	2	●	3	2	3	●	2	1	2	●	3	2	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
	1	u	1	1	2	4	●	s	a_1	a_2	a_3	t'	3	t	t'	3	●	●	2																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	1	2	2	1	1	●	●	4	●	3	2	3	●	2	1	1	2	t	●	2																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
	2	●	u'	2	2	4	s'	3	1	1	0	1	1	1	0	0	1	2	2	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
	2	●	●	3	u	u'	s	2	1	1	1	1	1	1	1	1	1	t'	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5: An AND gate on a normal Minesweeper

The reason that Figure 5 is actually an AND gate is as follows. If the result is T, then t must be T, which means that a_2, a_3 are both T. Then a_1 should be F, so s must be T, and by the symmetry of this logic gate, r also should be T. It implies that u' and v' are both F when we observe the blue square named '4'. Therefore, the result(r) is T only when two inputs(u and v) are both T, which concludes that this logic gate is actually an AND gate. The truth table for this AND gate is shown in table 2:

Table 2: Truth table for the AND gate

u	v	s	r	a_1	a_2	a_3	b_1	b_2	b_3	$t = u \wedge v$
T	T	T	T	F	T	T	F	T	T	T
T	F	T	T	T	F	T	T	F	T	F
F	T	T	T	T	F	T	T	F	T	F
F	F	F	F	T	T	F	T	T	F	F

Next figure is an OR gate made by Stefan [3]. Similarly we can easily check that the result(r) is F only when two inputs(u and v) are both F.

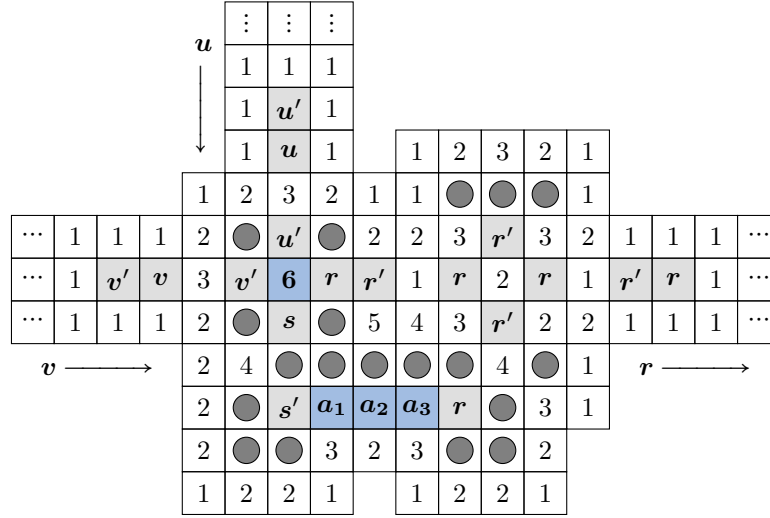


Figure 6: An OR gate on a normal Minesweeper

The truth table for this OR gate is as below:

Table 3: Truth table for the OR gate

u	v	s	a_1	a_2	a_3	$r = u \vee v$
T	T	T	T	T	F	T
T	F	F	T	F	T	T
F	T	F	T	F	T	T
F	F	F	F	T	T	F

3 Computational components in a hexagonal Minesweeper

As I discussed in the introduction, we can apply the computational components on a hexagonal grid as Bondt [1] made a hexagonal wire.

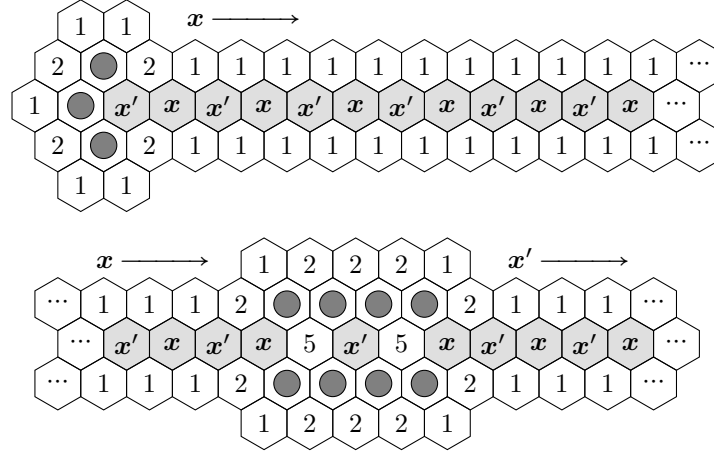


Figure 7: Hexagonal wire and a NOT gate [1]

Even though this application is only one of the possible forms of wires, the fact that we cannot distinguish ‘0’ and ‘1’ in an infinite wire without a starting point, supports the need for an improved form of a hexagonal wire [1]. The figure 8 below represents an improved form of wire on a hexagonal Minesweeper.

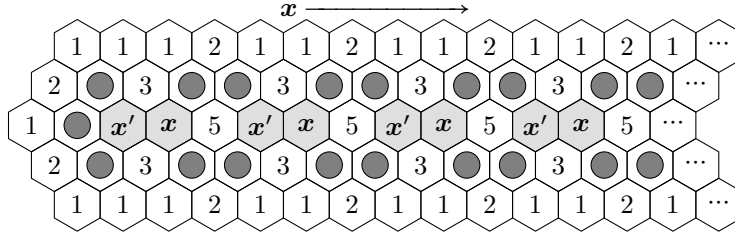


Figure 8: An improved form of a wire on a hexagonal Minesweeper

This form of wire allows us to distinguish x and x' clearly; in other words, it represents a phase of Boolean values. A NOT gate using this wire perfectly demonstrate *false* and *true* respectively. Of course, we can make a curve and a splitter of wires.



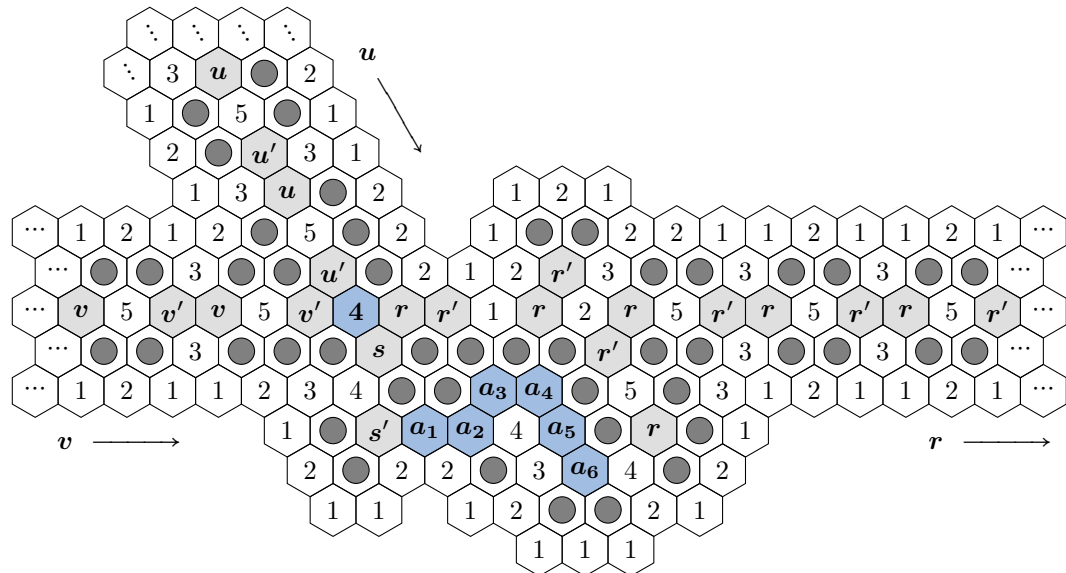
8



8

The diagram shows a hexagonal lattice with two sublattices, x and x' , indicated by arrows at the top. The lattice is divided into two sublattices, x and x' , which are represented by different shades of gray. The lattice is populated with various numerical labels: 1, 2, 3, 5, x , x' , and \dots . The labels are arranged in a pattern that suggests a specific physical or mathematical context, possibly related to a spin system or a lattice model. The labels 1, 2, 3, and 5 are placed in white hexagons, while x and x' are placed in shaded hexagons. Ellipses (\dots) are used to indicate that the lattice extends infinitely in all directions.

Duplicating two NOT gates, we can make a phase-changer easily (in this paper, I do not provide such figure as it is logically simple to determine). Now, I give an OR gate and an AND gate on a hexagonal grid. Figure 12 is an OR gate.



As the figure seems quite complicated, it requires some explanation for it being an OR gate. Let us prove this case-by-case.

Since \mathbf{u}' and \mathbf{v}' are both F, \mathbf{r} and \mathbf{s} should be both T. We can easily check that \mathbf{a}_1 , \mathbf{a}_3 , \mathbf{a}_4 , and \mathbf{a}_5 are T and the rest \mathbf{a}_i 's are F.

Investigating the blue '4', only one between \mathbf{r} and \mathbf{s} is T. If \mathbf{r} is F and \mathbf{s} is T,

then a_1 is T and a_2 is F. Now we are looking through a_1 to a_6 . Since a_2 is F, a_3, a_4, a_5 are all T. Therefore a_6 should be F. But then r should be T, which is a contradiction! Therefore r should be T and s is F. Then a_1 and a_6 are F and a_2, a_5 are T. Since both a_2, a_5 are T, only one between a_3, a_4 is T (we cannot decide which one should be T).

Case 3. (both u and v are F)

Since u' and v' are both T, r and s should be both F. We can easily check that a_2, a_3, a_4 , and a_6 are T and the rest a_i 's are F.

By examining the *Case 1* to *3*, we can finally conclude that figure 12 is actually an OR gate. The truth table for this OR gate is as below:

Table 4: Truth table for the OR gate on a hexagonal Minesweeper

u	v	s	a_1	a_2	a_3	a_4	a_5	a_6	$r = u \vee v$
T	T	T	T	F	T	T	T	F	T
T	F	F	F	T	TF or FT	T	T	F	T
F	T	F	F	T	TF or FT	T	T	F	T
F	F	F	F	T	T	T	F	T	F

By changing the position of a_1 through a_6 , we can make an AND gate similarly.

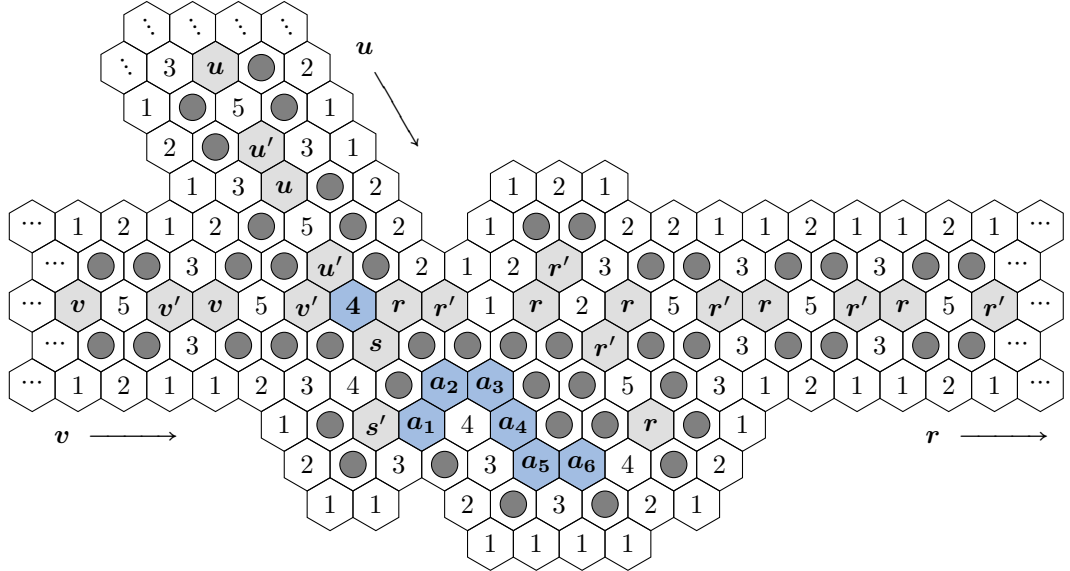


Figure 13: An AND gate on a hexagonal Minesweeper

The reason that figure 13 is actually an AND gate is very similar to that of the OR gate shown earlier in this paper. Notice that by examining through a_1

to a_6 , we can easily check that the result r is only T when two inputs u and v are both T. The truth table for this AND gate is as below:

Table 5: Truth table for the AND gate on a hexagonal Minesweeper

u	v	s	a_1	a_2	a_3	a_4	a_5	a_6	$r = u \wedge v$
T	T	T	T	T	T	F	T	F	T
T	F	T	T	TF or FT	T	T	F	T	F
F	T	T	T	TF or FT	T	T	F	T	F
F	F	F	F	T	T	T	F	T	F

4 PP-hardness of Minesweeper revisited

Bondt [1] showed that Minesweeper is PP-hard. First, he proved that weak MAJSAT is PP-complete, and then wired back the output of the circuit to the starting points of the inputs, in such a way that these inputs can be revealed when the output is known. In figure 14, I give such an example with an improved wire.

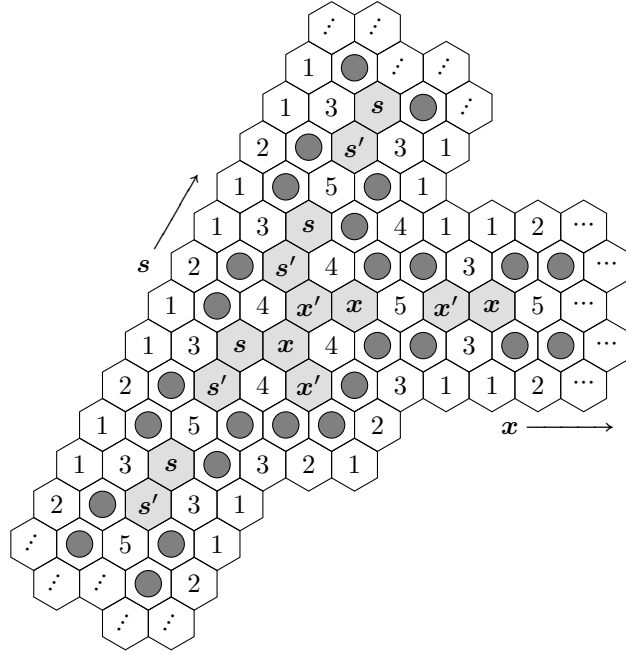


Figure 14: x can be revealed when s is known

Since \mathbf{x} is independent from \mathbf{s} , Bondt [1] rounded the number ϑ such that

$$\mathfrak{R}x_1\mathfrak{R}x_2\cdots\mathfrak{R}x_n\Pr[f(x_1,x_2,\cdots,x_n)]=\vartheta$$

to either 0 or 1, with a rounding error of at most 0.5. The symbol \mathfrak{R} means a random quantifier [4]. $\mathfrak{R}x_1$ implies a random choice, that is, the probability of true equals to 0.5 for true value for x_1 . As we have seen above, using an improved wire, we can also say that a hexagonal Minesweeper is PP-hard.

References

- [1] Michiel de Bondt, The computational complexity of Minesweeper, *arXiv:1204.4659v1 [cs.CC]*, 2012.
- [2] Richard Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer* **22**, nr. 2, pp. 9-15, 2000.
- [3] Richard Kaye, Some Minesweeper Configurations, *Portugese in Boletim Sociedade Portuguesa de Matemática, Janeiro(Número especial), Lisbon*. ISSN 0872-3672, pp. 181-189, 2007.
- [4] C. Papadimitriou, Games Against Nature, *J. Comput. System Sci.*, **31**, pp. 208-301, 1985.